Software-Vorgehensmodelle, Projektmanagement, Kosten-Zeitschätzung

Seminararbeit

im Rahmen der Projektgruppe „Ride.NET"
Jun.-Prof. Dr. Ralf H. Reussner,
Dipl. Wirtsch.-Inform. Steffen Becker
im WS 2004/2005 und SS 2005
Fk. II, Department für Informatik, Abt. Software Engineering
an der Carl-Von-Ossietzky Universität, Oldenburg

Klaus Krogmann

30. November 2004

Inhaltsverzeichnis

Zusammenfassung

Große Softwareprojekte stellen eine ernst zu nehmende Herausforderung dar, die mit den Mitteln des Software Engineerings (SE) bewältigt werden kann. SE bietet eine große Zahl von Softwareentwicklungsverfahren und Vorgehensmodellen an, die in die gleiche Richtung zielen, sich jedoch im Detail unterscheiden, weshalb eine genaue Kenntnis der Vor- und Nachteile von Nöten ist. Zudem unterliegt das SE einem permanenten Wandel, so dass beispielsweise das Wasserfallmodell zur Kontrolle der Softwareentwicklung im heutigen Umfeld nicht mehr zeitgemäß erscheint.

Dieser Text soll daher einen Überblick über die Bereiche Software-Vorgehensmodelle, Projektmanagement und Kosten-Zeitschätzung bieten und dabei aufzeigen, wo die Vor- und Nachteile spezifischer Methoden liegen. Im Zentrum der Betrachtung stehen dabei die am weitesten verbreiteten und akzeptierten Modelle vom Spiralmodell über den Bereich der Teamführung bis hin zur COCOMO Kostenschätzung.

1 Einleitung

Softwareprojekte werden tendenziell immer größer und bedeuten damit eine zunehmende Herausforderung für das Projektmanagement. Nach wie vor scheitert mit 26% ein nicht unerheblicher Prozentsatz von Projekten an schlechter Planung, wie [1] anführt.

Angesichts der vielfältigen Teilbereiche des Themas liefert der vorliegende Text vor allem einen Überblick über weit verbreitete und allgemein akzeptierte Methoden und Ansätze.

Im Bereich der Software-Vorgehensmodelle bietet das bis heute sehr weit verbreitete Wasserfallmodell einen klassischen Einstieg in die Vorgehensmodelle. Die neueren Modelle, wie die iterative Entwicklung und das Spiralmodell berücksichtigen die Anforderungen der modernen Softwareentwicklung.

Der Schwerpunkt im Bereich Projektmanagement liegt bei Empfehlungen für einen guten Umgang mit Teams, denn die Personalführung stellt einen sensiblen Punkt in der Softwareentwicklung dar.

Vor allem unerfahrenen Projektmanager bereitet die Kosten- und Zeitschätzung große Probleme, da eine Vielzahl von Parametern korrekt eingeschätzt werden müssen, um eine sinnvolle Prognose erstellen zu können. Das Function Point Verfahren und COCOMO stellen allgemein anerkannte Verfahren dar und werden daher in diesem Text behandelt.

2 Software-Vorgehensmodelle

Software-Vorgehensmodelle liefern eine abstrakte Sicht auf die Prozesse der Softwareentwicklung. Die Erfassung der Modelle umfasst die Rahmen der Prozessabschnitte, nicht jedoch ihre exakten Inhalte, wodurch sich die unterschiedlichen Softwareentwicklungs-Paradigmen einfacher vergleichen lassen. Speziell bei größeren Projekten wird nicht nur ein einzelnes Vorgehensmodell verfolgt, sondern, abhängig von den zu entwickelnden Teilsystemen, werden die jeweils passendsten Modelle angewendet.

Neben Prozessen der Klasse des Wasserfallmodells und der spiralförmigen Entwicklung findet auch die Prozessklasse der *formalen Systementwicklungsmethoden* Anwendung. Die formale Systementwicklung setzt eine Spezifikation der Anforderungen in einer mathematischen Notation voraus. Ihre Vorteile liegen in der Möglichkeit aus der Systemspezifikation direkt ein ausführbares Programm zu erzeugen. Auch Entwurf, Implementierung usw. stellen einen Umwandlungsprozess dar, bei dem die mathematische Repräsentation des Systems immer weiter detailliert wird. Da der gesamte Prozessverlauf verifiziert werden kann, ist es möglich, die Implementierung der Spezifikation im Programm formal sicherzustellen. Der große Aufwand für die Verifikation ist aber zugleich das Manko dieser Methode, womit sie sich schlecht für große Projekte eignet und nur dort Sinn macht, wo die Korrektheit eines System wichtig ist (vgl. [8], S. 57ff).

2.1 Wasserfallmodell

Anhand des weit verbreiteten Wasserfallmodells sollen im Folgenden vor allem die typischen Phasen der Softwareentwicklung aufgezeigt werden (vgl. [8], S. 57f).

Analyse und Definition der Anforderungen In Zusammenarbeit mit dem Kunden werden die Dienste, Ziele und Einschränkungen an das zu entwickelnde System aufgestellt. Die anschließende detailliertere Definition dient abschließend als Systemspezifikation.

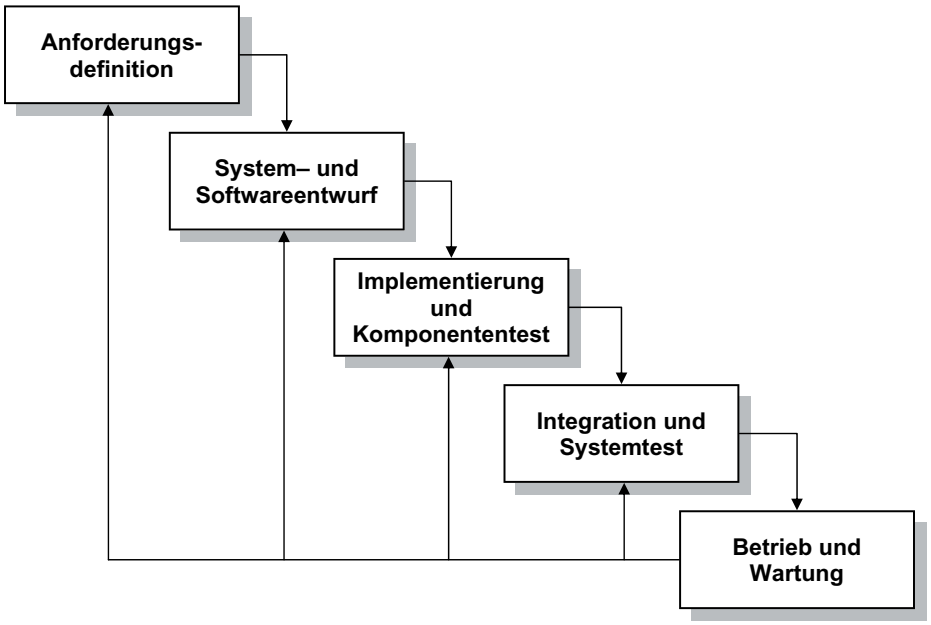

Abbildung 1: Das Wasserfallmodell (aus [8], S. 58)

System- und Softwareentwurf Im Systementwurf wird eine allgemeine Systemarchitektur festgelegt und eine Zuordnung von Anforderungen an Soft- bzw. Hardware vorgenommen. Der Softwareentwurf beinhaltet dem gegenüber die Erfassung der erwarteten Systemteile und ihrer Interaktion untereinander.

Implementierung und Komponententest Die im Softwareentwurf erfassten Systemteile werden jeweils in (einer Menge von) Programmteilen umgesetzt. Über den Test der Programmteile wird die Einhaltung ihrer Spezifikation sichergestellt.

Integration und Systemtest Dieser Schritt umfasst den Test aller Teilkomponenten als Gesamtsystem. Dazu werden alle Teile integriert und gegen die Softwareanforderung getestet. Verlaufen die Tests zufriedenstellend, erfolgt die Auslieferung des Produkts an den Kunden.

Betrieb und Wartung In dieser Phase wird die Software zunächst einmal in Betrieb genommen (installiert) und für die Benutzer eingerichtet. Treten im Laufe des Betriebs noch nicht erkannte Fehler auf, werden diese korrigiert. Ebenso werden neue Anforderungen und Erweiterungen über Veränderungen der Implementierung umgesetzt.

Typischerweise wird jeder Prozesschritt mit der Anfertigung von Dokumenten begleitet. Der Eintritt in eine nächste Prozessphase erfolgt diskret, also nicht vor Abschluß einer vorangehenden. Damit ist das Wasserfallmodell einfach zu überwachen und verwalten.

Wie Abbildung 1 erfasst, werden in der Praxis jedoch häufig Rückkopplungen zwischen den verschiedenen Phasen vorgenommen. Treten in frühen Phasen Fehler auf, ziehen diese in allen späteren Prozessabschnitten Probleme nach sich (vgl. Kapitel 4). Durch Rückschritte im Prozess können Fehler jedoch frühzeitig korrigiert werden. Werden dabei

auf Grund des damit verbundenen Aufwands nicht alle Prozesschritte des Wasserfallmodells von Beginn an vollständig durchlaufen, kann dies ebenfalls zu Fehlern führen, da eher Workarounds, denn Problemlösungen durchgeführt werden[1]. In der Folge ist das System schlecht implementiert und führt zu weiteren Problemen bei Betrieb und Wartung.

Das Hauptproblem des Wasserfallmodells ist die Aufteilung in starre Prozessabschnitte, ohne dass eine Revision früherer Phasen zu späteren Zeitpunkten vorgesehen ist. Dies bedeutet, dass alle Anforderungen prinzipiell von Beginn an exakt feststehen müssen. Wie auch Kapitel 4 hervorhebt, ist dies zumeist nur schwerlich möglich.

2.2 Spiralförmige und inkrementelle Entwicklung

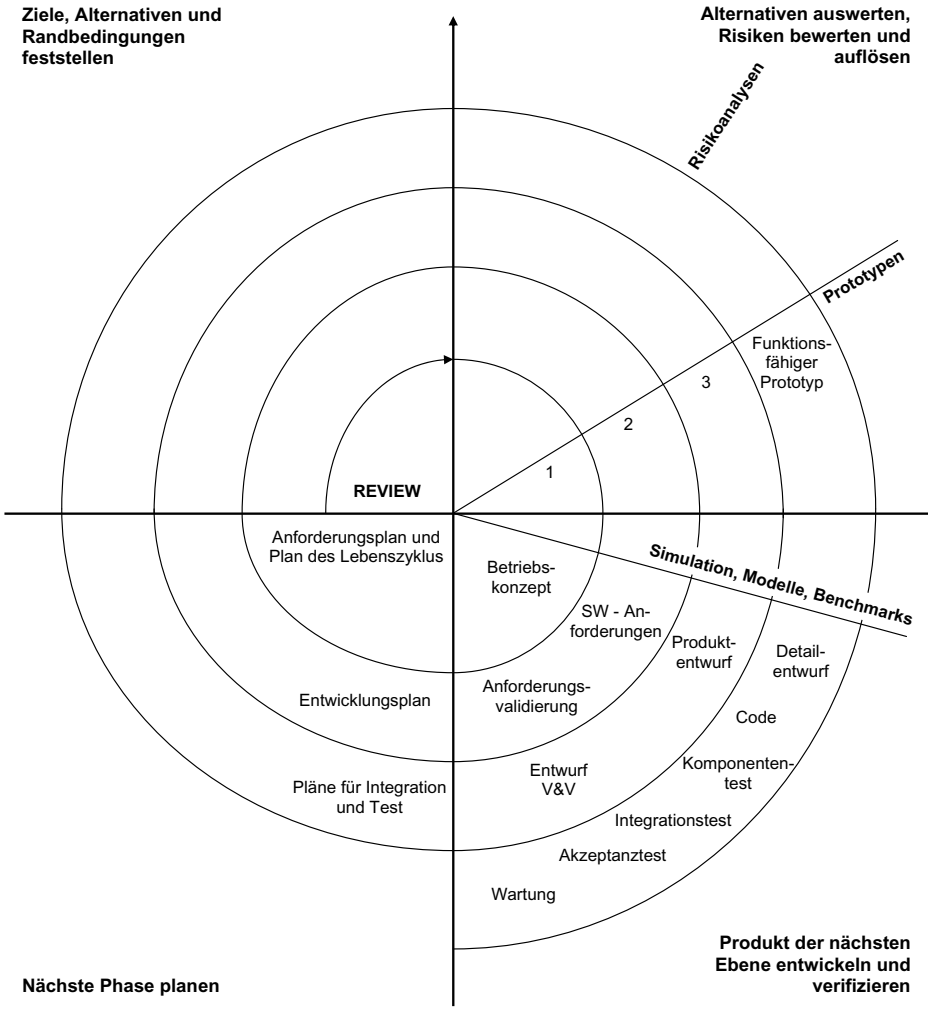

Abbildung 2: Spiralmodell nach Boehm (© 1988 IEEE, aus [8])

Die spiralförmige (siehe Abbildung 2) wie auch die inkrementelle Entwicklung berücksichtigen, im Gegensatz zu z. B. dem Wasserfallmodell, wiederholende Entwicklungsaktivitäten, wodurch bereits durchgeführte Arbeiten wie Entwurf und Implementierung

[1] bspw. die Umsetzung einer Funktion auf Grund eines vorangehenden Fehlers anders als in der Anforderung, ohne eine Korrektur der Anforderungen, wodurch die Kundenbedürfnisse nicht mehr erfüllt werden.

nicht immer wieder vollständig durchgeführt werden müssen. Dabei wird die Spezifikation zusammen mit der Software entwickelt. Dadurch fehlt allerdings bei vielen klassischen Geschäften eine ausreichende Grundlage für den Vertragsabschluß zwischen Kunde und Softwarehersteller, was zu Akzeptanzproblemen führen kann. Je nach Projektphase ist der Einsatz bestehender Entwicklungmodelle wie dem Wasserfallmodell oder der formalen Systementwicklung ausdrücklich vorgesehen.

Die *inkrementelle Entwicklung* stellt eine Mischung aus dem Wasserfallmodell und der evolutionären Entwicklung (siehe [8], S. 58f) dar und versucht beide Ansätze optimal zu kombinieren. Die Kernidee besteht darin, dass das System zunächst grob in Teilsysteme zerlegt wird. Entsprechend der Priorität ihrer Funktionen werden die Teilsysteme Stück für Stück entwickelt und erst zu diesem Zeitpunkt werden ihre Details festgelegt. Alle bereits fertiggestellten Teile können vom Kunden eingesetzt werden. Durch diese Einsatz-Tests können bereits früh neue Anforderungen an die verbleibenden Teile ermittelt und direkt umgesetzt werden. Damit ist auch ein Fehlschlagen von Projekten unwahrscheinlicher, da Kundenanforderungen direkt umgesetzt werden können, Kernkomponenten gut getestet sind und Probleme in einem der Teilsystemen nicht zwangsläufig Auswirkungen auf das Gesamtsystem haben.

Bei der *spiralförmigen Entwicklung* werden die Phasen des Softwareentwicklungsprozesses als Spiralebenen dargestellt. Von innen nach außen werden die einzelnen Entwicklungsstufen, wie Anforderungsdefinition, Systementwurf usw. durchlaufen. Zudem erfolgt eine Aufteilung aller Ebenen in vier Segmente, die für jede Ebene durchlaufen werden:

Ziele festlegen Für die aktuelle Phase werden neue Projektziele und deren Alternativen erfasst, und die Bedingungen unter denen diese erreicht werden müssen, festgestellt. In diesem Segment wird außerdem das Projektmanagement für den nächsten Zyklus unter Berücksichtigung der erkennbaren Risiken durchgeführt.

Risikobewertung Das Spiralmodell ist ausdrücklich risikobetont und verlangt daher in diesem Segment eine Analyse der Projektunsicherheiten (zur Risikoabschätzung siehe auch Kapitel 4.3) und möglicher Wege zur Verringerung des Risikos, wie die Erstellung von Prototypen oder die Durchführung von Simulationen.

Entwicklung und Validierung Auf Basis der Ergebnisse der Risikobewertung wird ein möglichst passendes Entwicklungsmodell für den nächsten Prozessschritt gewählt um die Hauptrisiken zu minimieren (zur Wahl geeigneter Modelle siehe auch [8], S. 65).

Planung Wird in diesem Segment entschieden, dass es für das Projekt sinnvoll ist, mit der nächsten Spiralebene fortzufahren, werden die Planungen für die nächste Phase angestoßen.

Da das Spiralmodell andere Prozessmodelle aufgreift und verwendet, gibt es keine festen Phasen wie Entwurf oder Spezifikation. Dadurch ergibt sich aber auch die Möglichkeit, je nach der Art der Anforderungen an eine Phase ein geeignetes Prozessmodell wählen zu können, das die Risiken am Besten minimiert. So kann beispielsweise eine formale Systemspezifikation zur Erfüllung von Sicherheitsanforderungen einem „Wasserfallzyklus“ folgen. Damit bietet, die spiralförmige Entwicklung die Möglichkeit die Vorteile bekannter Prozesse zu vereinen.

3 Projektmanagement

Zu den Aufgaben des Projektmanagements zählen klassischerweise (nach [8], Seite 84f):

Angebotserstellung Die Angebotserstellung stellt den Ausgangspunkt für ein Softwareprojekt dar. Im Angebot werden die Durchführung und Ziele des Projekts beschrieben. Dies umfasst ebenfalls Abschätzungen über die zu erwartenden Kosten und den Zeitaufwand (siehe auch Kapitel 4) des Projekts. Da es in der Wirtschaft die Grundlage für den Abschluss von Verträgen bildet, sollten hierin die besonderen Stärken und Herausstellungsmerkmale des anbietenden Unternehmens bzw. Teams dargestellt werden.

Projekt- und Zeitplanung Die Projektplanung umfasst die Festlegung von Meilensteinen, Aufgabenbereichen und Release-Versionen. Eine Teilaufgabe ist ensprechend die Festlegung eines Plans (siehe auch Kapitel 3.1.1), der die Entwicklungsschritte festhält. Daran schließt sich die Zeitplanung an, die eine Abschätzung der benötigten Ressourcen zur Realisierung von Teilaufgaben vornimmt (siehe auch Kapitel 4). Die Projektkostenkalkulation übernimmt schließlich die Prognose der dabei entstehenden Kosten.

Projektüberwachung und Reviews Während die Projektüberwachung permanent (durch das Projektmanagement) während eines Projekts greift, finden Reviews punktuell statt und beziehen auch externe Anspruchsgruppen ein. Darin wird die Bewertung des Gesamtfortschritts des Projekts, des technischen Entwicklungsstands und des Projektstatus anhand des Projektplans vorgenommen. Neben der Entscheidung über den Einsatz eines (möglicherweise externen) Experten bei Problemen, obliegt dem Projektmanager die Aufgabe, insbesondere bei langwierigen Projekten, einen Zielabgleich vorzunehmen, also zu überprüfen, ob die ursprünglichen Ziele des Projekts immer noch Bestand haben, und bei Bedarf Anpassungen vorzunehmen.

Personalauswahl und -Beurteilung Im Idealfall sollte ein Projektmanager nur diejenigen Menschen auswählen, die über die größtmöglichen Erfahrung und Fähigkeiten in ihren Projektbereichen verfügen. Da qualifiziertere Teammitglieder jedoch teurer sind, das Budget üblicherweise jedoch begrenzt ist, müssen Kompromisse eingegangen werden. Dies ist auch der Fall, wenn der Projektmanager keinen oder nur stark begrenzten Einfluß auf die Auswahl des Teams hat. Dennoch sollte wenigstens *ein* erfahrenes Mitglied im Team vorhanden sein, um bei größeren Problemen steuernd wirken zu können (siehe hierzu auch Kapitel 3.1).

Berichterstellung und -Präsentation Neben der Angebotserstellung und möglichen Reviews tritt das Projektmanagement mit dem Kunden in Form von Berichten in Kontakt. Daneben informieren die knappen Projektberichte auch intern z. B. das Unternehmen über den Projektfortschritt. Der Projektmanager übernimmt in mündlicher und schriftlicher Form eine vermittelnde Rolle.

3.1 Teams

Die Ausführungen über Teams in diesem Kapitel beziehen sich vor allem auf den TSPi-Prozess, wie er in [3] beschrieben wird, haben aber auch außerhalb des TSPi eine allge-

meine Bedeutung für den Softwareentwicklungsprozess, wie zum Beispiel [8] im Kapitel „Personalmanagement“ (Seite 503ff) beschreibt.

Der menschliche Einfluß stellt demzufolge den wichtigsten Faktor in der Softwareentwicklung dar. Da das Kapital einer Organisation, die Software entwickelt, üblicherweise immaterieller Natur ist und vor allem aus dem Wissen seiner Mitarbeiter besteht, bilden Entwicklungsteams den wichtigsten Produktionsfaktor. Entsprechend bedeutet ein gutes Projektmanagement auch, dass Lösungen von technischen Problemen durch möglichst effektive Delegationen von Personal erreicht werden müssen.

3.1.1 Teammanagement

Die meisten Softwareprojekte schlagen nicht fehl, weil die technischen Ansprüche an eine Software nicht erfüllt werden, sondern weil die Teamarbeit fehlschlägt, wie [3] anführt. Die Teammitglieder werden beispielsweise in einem zu straffen Entwicklungsprozess verplant, der dadurch einen Druck auf sie aufbaut. In der Folge führt der Druck zu typischen Ausweichmechanismen, mit denen die Teammitglieder reagieren. Dazu zählen beispielsweise Abkürzungen, wie sie schlecht implementierte Methoden bieten oder das bloße „Herumspielen“ mit neuen Enwicklungstechniken, Werkzeugen oder Sprachen, anstelle einer ernsthaften Auseinandersetzung mit den anstehenden Problemen. In der Folge sinkt die Produktivität weiter, trotz des erhöhten Drucks.

Daraus ergeben sich direkt Anforderungen an die Organisation eines Entwicklungsprozesses, also an den Projektmanager. Ein zu hoher Druck auf die Individuen eines Teams wirkt eher destruktiv, da Probleme falsch eingeschätzt werden, Probleme konstruiert werden, die in der ausgemalten Weise nicht real sind, zu Unmut führen und somit Einfluss auf das Problemlösungsvermögen nehmen, wie [3] ausführt.

Ein sinnvoller Weg, um einen kontrollierten Druck auszuüben, der eher einen weisenden Charakter hat, ist die Aufstellung eines Plans zu Beginn eines Projekts. Der Plan sollte dabei genau erfassen, welche (humanen) Ressourcen von einzelnen Abschnitten betroffen sind. Selbst bei großen, zunächst unüberschaubaren Projekten, sind Teilproblemlösungen absehbar, die in der Folge zu einer kontrollierten Arbeitsatmosphäre führen und somit den Druck auf Einzelpersonen auf ein angenehmes Maß reduzieren, das gleichwohl die Produktivität erhalten.

Ebenso erstreckt sich die Führungsaufgabe des Projektmanagements auf die Motivation von Mitarbeitern sowie die Sicherstellung, dass die avisierten Ziele tatsächlich erfüllt werden und die Qualität der durchgeführten Arbeit den Ansprüchen genügt.

Das Aufstellen eines Projektplans darf daher als ein Mittel gesehen werden, Teammanagement zu betreiben. Ziel des Projektmanagements ist somit auch die Optimierung des Projekterfolgs über das Teammanagement.

3.1.2 Teamprobleme

Für die meisten Probleme in Teams lassen sich wiederkehrende Ursachen (vgl. [3], Seite 17ff) ausmachen, nach deren Lösung die Vorteile der Teamarbeit zum Tragen kommen.

Führung Die Führung eines Teams ist häufig ineffektiv und führt dadurch zu Problemen bei der Umsetzung der gegebenen Pläne und zur Verringerung der Arbeitsdisziplin. Dabei ist wichtig zu beachten, dass zwischen einer Fach- und Persönlichkeitsautorität (über Ausstrahlung) und einer Positionsautorität (vgl. [9], S. 193ff) unterschieden werden

muss. Die Fach- und Persönlichkeitsautorität entsteht zumeist auch unter gleichberechtigten Teammitgliedern durch die Bündelung von Fach- und Führungskompetenz in einer Person. Dem gegenüber kann eine Positionsautorität, die auf Grund der Organisationsstruktur eine Führungsrolle samt Sanktionsgewalt innehat, weniger Fähigkeiten und Übung in der Führung von Personen haben.

Daraus ergibt sich, dass die Fach- und Persönlichkeitsautorität und die Positionsautorität nicht in einer Person vereint sein müssen. Um in diesen Fällen Konflikte zu vermeiden, bieten sich zwei Lösungswege an: Die Positionsautorität sollte gut geschult werden um auch die Fachautorität zu sein, oder die Positionsautorität sollte trivialerweise bereits die Fachautorität sein.

Kooperation Der mangelnde Wille einzelner Mitglieder kooperativ im Team zu arbeiten, kann die Teamarbeit ernsthaft gefährden. Das wirksamste Mittel zum Abstellen solchen Verhaltens ist der Druck auf das betreffende Teammitglied durch Gleichgestellte. Sollte dies immer noch keine Abhilfe schaffen, sind Interventionen durch die Teamführung anzudenken, die auch disziplinarische Maßnahmen einschließen können.

In die gleiche Problemrichtung geht die mangelnde Beteiligung einzelner Personen an der Mitarbeit. Dies führt zwangsläufig zu einer ungleichen Verteilung der Arbeit. Innerhalb der Gruppe entstehen somit Benachteiligungen anderer, was wiederum die Atmosphäre im Team trüben kann. In gleicher Weise können weitere Teammitglieder ihren mangelnden Willen zu hohem Engagement begründen. Dies führt letztlich zu einem Motivationseinbruch im Team und somit zu Ineffektivität. Daher sollte die Arbeitsbelastung der einzelnen Teammitglieder bereits während der Planung möglichst (auch für andere erkennbar) gleich hoch sein.

Meilensteine Fehlende Meilensteine führen dazu, dass Probleme nicht gelöst, sondern vermehrt verschoben werden. Dies geht zu Lasten der Leistungsfähigkeit und führt dazu, dass zunächst vermehrt „beliebte“ Probleme behandelt werden, unabhängig davon, ob ihre Lösung in der Zeitlinie sinnvoll ist. Werden Meilensteine auf der anderen Seite unrealistisch gesetzt und daher nicht erreicht, führt dies zur Demotivation der Teammitglieder und weniger Vertrauen in die eigenen Fähigkeiten. In beiden Fällen liegt der Fehler vor allem in unerfahrenen Führern, die bspw. einen ähnlichen Entwicklungsprozess noch nicht begleitet haben. Auf der anderen Seite kann aber auch eine ungenaue Beschreibung der Ziele (im Sinne der Anforderungsdefinition, vgl. Kapitel 2.1) Grund für die Probleme sein. „Planvolle“ Meilensteine sind also für ein Projekt sinnvoll.

Qualität Verbesserungen der Qualität von Produkten lassen sich sehr gut auf der Teamebene erreichen. Durch Review-Sitzungen und Peer-Evaluationen lässt sich die Qualität signifikant steigern (vgl. auch [6], Seite 302), sofern die Bemühungen um Verbesserungen in allen Bereichen der Arbeit greifen und nicht z. B. nur zu einem exzellenten Entwurf, ansonsten aber mäßiger Dokumentation und Implementierung usw. führen. Dabei ist zu beachten, dass gerade Peer-Beurteilungen häufig auf Widerstreben stoßen und daher nicht immer vollständig ehrlich durchgeführt werden. In der Folge kann das Gefühl mangelnder Fairness das Gerechtigkeitsempfinden im Team stören. Daher sollte die Führung von Beginn an darauf dringen, das Bewertungen stets offen und aufrichtig sein sollten.

3.1.3 Teameffizienz

Nachdem die häufigen Probleme im Teamumfeld skizziert wurden, verbleibt die Betrachtung der optimalen Wahl eines geeigneten Teams. Wie [3] (Seite 19f) ausführt, liegt in den meisten Fällen die optimale Teamgröße bei sechs Personen. Eine Erhöhung der Größe verringert die Effizienz, während kleinere Gruppen Probleme bei einer eindeutigen Rollenverteilung haben. Die Größe ergibt sich dabei aus der Möglichkeit innerhalb kleiner Gruppen, dass sich alle Personen sehr gut kennen können und genau wissen, welche Aufgaben von welchem Teammitglied übernommen werden. Die Mitglieder können voneinander lernen.

Bei großen Gruppen (vor allem ab 12 Personen) hingegen nimmt der Kommunikationsoverhead deutlich zu. Das Entwickeln von Beziehungen innerhalb der Gruppe wird erschwert. Zudem wird die Rollenverteilung undurchsichtiger, wodurch die Verantwortlichkeiten jedes Teammitglieds unklarer werden.

Teams gelten dann als effizient, wenn sich alle Mitglieder ihrer (gemeinsamen) Aufgabe bewußt sind und diese im Teamwork selbstverwaltet ausführen. Die häufige Kommunikation (z. B. auf Team-Treffen) innerhalb der Gruppe sollte offen geschehen. Im Idealfall handelt ein Team als Einheit - mit gemeinsamen Werten und Zielen. Durch das Anstreben konkreter meß- und bewertbarer Ziele wie Performance-Marken, Qualitätsmaßstäben und geplanten Meilensteinen, läßt sich dann eine Erhöhung der Motivation erreichen, da sich ein (positiver) Wettbewerb mit klaren Regeln innerhalb des Teams entwickeln kann. Die Fortschritte in Richtung des definierten Ziels werden somit bewußt erfahrbar.

In Erweiterung zu [3] ergeben sich für erfolgreiche Teamarbeit somit folgende grundsätzliche Rahmenbedingungen, die jederzeit klar sein müssen (vgl. auch [5], Seite 298):

- Was ist das Gesamtziel des Projekts?
- Welche Aufgaben müssen dazu gemacht werden?
- Wann und in welcher Reihenfolge?
- Wer hat welche Aufgaben und Fähigkeiten?
- Wo stehen wir?

3.2 Projektverwaltung

3.2.1 Zieldefinition

Wie bereits in Kapitel 3.1 gezeigt wurde, ist die Festlegung von Zielen ein wichtiges Instrument zur Führung eines Teams, dass als „Management by Objectives“ bezeichnet wird (vgl. [9]). In diesem Kapitel wird aufgezeigt, wie dieses aussehen kann.

Ziele sollten zunächst konkret benannt und beschrieben werden. Im nächsten Schritt sollten zu jedem Ziel Erreichungsgrade festgelegt werden, wie das Beispiel zeigt (vgl. [3], Seite 31):

Ziel 1: Erstellen eines qualitativ hochwertigen Produkts

Maß 1.1: Prozentzahl der gefundenen Fehler vor dem ersten Compilieren: $> 80\%$

Maß 1.2: Anzahl der gefundenen Fehler im Testsystem: 0

Maß 1.3: Anforderungsfunktionen die bei Projektfertigstellung vorliegen: 100%

Ziel 2: Durchführung eines produktiven Projektverlaufs

Maß 2.1: Abweichungen von der geplanten Produktgröße: $< 20\%$
Maß 2.2: ...

Durch ihre Syntax ist die QML (Quality Markup Language, vgl. [2]) in abgewandelter Form ebenfalls zur Beschreibung von Zielerreichungsgraden nutzbar: die Definition des Vertragstyps enthielte die Zielbeschreibung im Typ, der Vertrag dazu enthielte dann die Metriken, ein konkreter Vertrag dann die Erreichungsgrade in verschiedenen Abstufungen.

3.2.2 Aufgabenverteilung

Um jedem Teammitglied einen Überblick über seine persönlichen Aufgaben im Team zu verschaffen, bietet es sich an, wie in Kapitel 3.2.1 beschrieben, ebenfalls Zieldefinitionen für die Teilaufgaben zu verfassen. Jedem Teammitglied wird dadurch ein bestimmter Verantwortungsbereich übertragen, der von allen einsehbar sein sollte (etwa über eine tabellarische Auflistung der Verantwortungsbereiche, vgl. [3], Seite 34f), damit mögliche Probleme und Fragen direkt mit der entsprechenden Person geklärt werden können.

4 Kosten-Zeitschätzung

Ein wichtiger Bestandteil des Projektmanagements ist, wie bereits in Kapitel 3 angedeutet, die Kosten- und Zeitschätzung von Projekten. Zur Kosten- und Zeitschätzung haben sich grafische Notationen und Analysemethoden herausgebildet, die zur Abschätzung der Aufwände besonders empfehlenswert sind und im Folgenden näher betrachtet werden.

4.1 Projektzeitplan

Der Projektzeitplan erfasst die Zeiten, Ressourcen und Aufgaben, die für Abschnitte eines Projekts bis zu seiner Fertigstellung benötigt werden. Da gerade in der Softwareentwicklung häufig sehr unterschiedliche Projekte abgewickelt werden, liegen in den meisten Fällen keine Erfahrungswerte über den zu erwartenden Aufwand vor. Entsprechend sind alle Planungen nur grobe Schätzungen des tatsächlich zu erwartenden Aufwands.

Insbesondere (technische) Probleme lassen sich im Vorfeld nicht abschätzen und führen somit häufig zu Verzögerungen. Gerade in großen Projekten werden garantiert Probleme auftreten. Daher sollten alle Abschätzungen ausreichend Puffer zum Abfangen von unerwarteten Ereignisssen enthalten, um dem Projektzeitplan überhaupt eine realistische Komponente zu geben. Daraus resultiert ebenfalls die Anforderung, dass der Projektplan stetig an den aktuellen Entwicklungsstand angepasst werden muss.

Ein Projektplan sollte die möglichst effiziente Verteilung der zur Verfügung stehenden Ressourcen realisieren, also auch parallel durchgeführte Aufgaben derart koordinieren, dass sich möglichst wenig zeitkritische Abhängigkeiten ergeben. Als Ressource gilt in diesem Zusammenhang vor allem die menschliche Arbeitskraft. Daneben sind aber auch begrenzte Hardware- / Serverkapazitäten und das Budget als Hauptressourcen zu sehen, die optimal genutzt werden sollten, bei Ausfall aber den Projektplan möglichst wenig in Verzug bringen.

Wie [8] ausführt, empfiehlt es sich, 30% der Zeit für die Behandlung erwarteter Probleme und weitere 20% für die Behandlung nicht erwarteter Probleme als Puffer einzuplanen.

4.2 Visualisierung

Um den Entwurf des Projektplans zu vereinfachen werden zumeist Diagrammdarstellungen verwendet, die sich z. B. mit Microsoft Project erstellen lassen. Darin werden (je nach Typ) Zuordnungen von Verantwortlichkeiten für Aufgaben vorgenommen, Abhängigkeiten zwischen Teilaufgaben dargestellt und Anfangs- bzw. Endzeiten von Teilabschnitten festgehalten.

4.2.1 Abhängigkeitendarstellung

Nr.		Vorgangsname	Dauer	Anfang	Ende	Vorgänger
1		T1	2 Tage	Mon 27.09.04	Die 28.09.04	
2		T2	5 Tage	Mit 29.09.04	Die 05.10.04	1
3		T3	2 Tage	Mit 06.10.04	Don 07.10.04	2;4
4		T4	3 Tage?	Mit 29.09.04	Fre 01.10.04	1
5		T5	1 Tag?	Fre 08.10.04	Fre 08.10.04	3
6		T6	4 Tage	Mit 06.10.04	Mon 11.10.04	4
7		T7	3 Tage	Die 12.10.04	Don 14.10.04	6
8		T8	2 Tage	Fre 15.10.04	Mon 18.10.04	7
9		T9	4,5 Tage?	Mon 27.09.04	Fre 01.10.04	
10		T10	1 Tag?	Mon 27.09.04	Mon 27.09.04	

Abbildung 3: Abhängigkeitsdarstellung, MS Project

Die Darstellung der Teilaufgaben, Abhängigkeiten und benötigten Zeitdauern wird in Abbildung 3 exemplarisch vorgenommen. Zur Erstellung dieser, wie auch der weiteren Beispielgrafiken zu diesem Thema wurde Microsoft Project 2003 verwendet. Angesichts des Umfangs der Software, werden hier nur einige Grundfunktionen tatsächlich verwendet.

Jede Aufgabe (Vorgang) bekommt zunächst einen beschreibenden Namen, wie etwa „Dokumentationsaktualisierung“. Im Beispiel wurden zur Kennzeichnung T1 bis T10 verwendet. Außerdem wird zu jeder Aufgabe die erwartete Dauer festgehalten. Ein Fragezeichen hinter einer Angabe (etwa bei T4) kennzeichnet, dass die Schätzung des Zeitaufwands unsicher ist. Daneben verwaltet MS Project eine konrete Datumsangabe zu den Anfangs- und Endzeiten. Die Abhängigkeitendarstellung einzelner Aufgaben von anderen erfolgt in Listenform („Vorgänger“). Im Beispiel hängt T2 von T1 ab, so dass der Anfangszeitpunkt von T2 nicht vor dem Ende von T1 liegt.

Bei der Modellierung sollte darauf geachtet werden, dass die Definition von Teilaufgaben nicht zu kleinteilig erfolgt. Typische Aufgaben sollten (entgegen der Beispieldarstellung) mindestens einen Zeitraum von einer Woche umfassen, damit der Verwaltungsoverhead für den Projektplan bei vielen kleinen Änderungen nicht übermäßig groß wird.

4.2.2 Balkendiagramm

Die Balkendiagramme (siehe Abbildung 4) in MS Project werden direkt aus den spezifizierten Abhängigkeiten erzeugt. Die Breite der Balken zeigt dabei die Dauer der Aufgaben an. Durch die Einordnung der Balken in ein Kalenderraster werden die Anfangs-

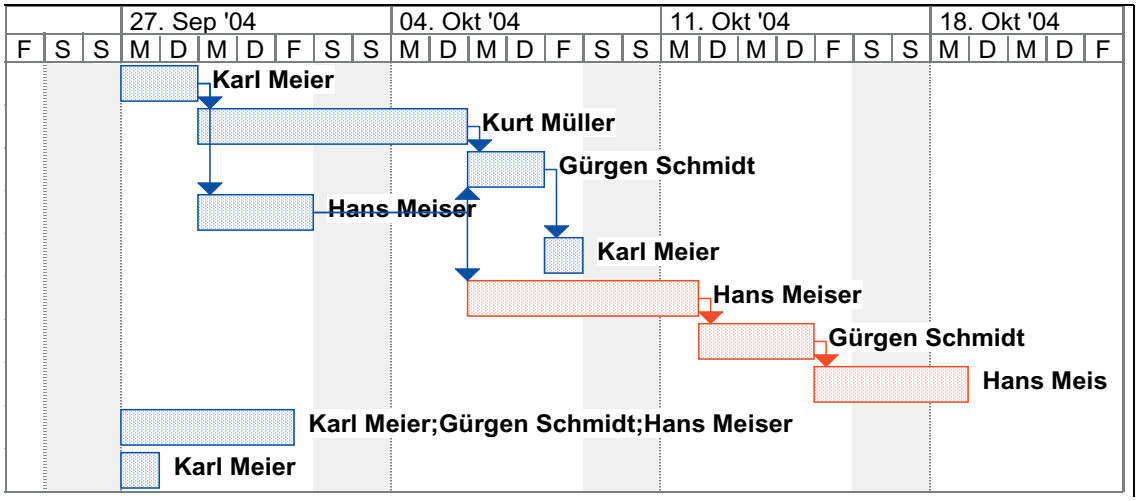

Abbildung 4: Balkendiagramm: Gantt-Chart

und Enddaten visualisiert. Zur Unterstützung des Projektmanagements lassen sich kritische Aufgaben in einer gesonderten Farbe (hier rot) darstellen. Damit ist erkennbar, welche Verzögerungen einer Aufgabe zu besonders großen Verschiebungen des Gesamtplans führen. Die Abhängigkeiten von Teilaufgaben untereinander werden durch Pfeile angezeigt. So weist der zweite Balken (T2) von oben eine Abhängigkeit vom ersten Balken (T1) auf.

Daneben erfasst MS Project in den Balkendiagrammen auch die Verwendung bzw. Zuordnung von Ressourcen zu Aufgaben, wie die Namen rechts der Balken im Beispiel andeuten. MS Project akzeptiert z. B. auch mehrere Ressourcen pro Aufgabe („Meier, Schmidt, Meiser“), sowie eine anteilige Zuordnung (Bsp.: „Schmidt 30%“) einer Ressource zu einer Aufgabe. Damit lässt sich eine Priorisierung von Ressourcen vornehmen.

Sowohl das Balken- als auch das Netzdiagramm (siehe Kapitel 4.2.3) zeigen auf, welche Arbeiten parallel durchgeführt werden können und bieten durch ihre Darstellungsform eine Möglichkeit den Projektplan bezüglich der Auslastung spezieller Ressourcen, der Gesamtdauer oder anderer Ziele zu optimieren, da Projektabhängigkeiten erkennbar werden, die nicht intuitiv zu erfassen sind.

4.2.3 Netzdiagramm

Auch das Netzdiagramm (Abbildung 5; auch „Netzplan“, vgl. [8]) basiert auf den Daten aus der Abhängigkeitsspezifikation. Jede Aufgabe wird als einzelner Kasten dargestellt, der die belegte(n) Ressource(n) und die erwartete Dauer enthält. Netzdiagramme werden im Allgemeinen von links oben nach rechts unten gelesen. Eine Aufgabe kann gestartet werden, wenn alle davor liegenden Aufgaben (entgegen der Pfeilrichtung) erfüllt wurden. Durch Addieren der Dauern in Pfeilrichtung kann die prognostizierte Zeit bis zum Erreichen einer bestimmten Aufgabe abgelesen werden.

In der gleichen Weise kann auch der so genannte kritische Pfad ermittelt werden, also die als minimal erwartete Zeit zur Durchführung eines Projekts. Verzögerungen auf dem kritischen Pfad führen direkt zur Verzögerung des Gesamtprojekts, weshalb der kritische Pfad oder kritische Teilabschnitte für das Projektmanagement von besonderem Interesse sind. Gelingt es Verkürzungen des kritischen Pfads zu realisieren, wirkt sich dies um-

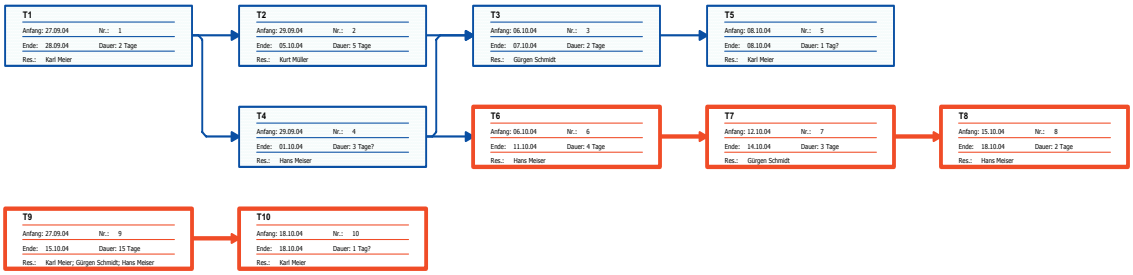

Abbildung 5: Netzdiagramm

gekehrt direkt auf die Projektdauer aus, womit unter Umständen beispielsweise Kosten gesenkt werden können. MS Project zeigt den kritischen Pfad im Form von roten Kästchen an.

Verzögerungen auf Abschnitten, die nicht zum kritischen Pfad gehören, sind so lange ohne Bedeutung für die Gesamtdauer des Projekts, wie sie die Dauer des entsprechend parallel verlaufenden kritischen Teilpfads nicht überschreiten.

4.2.4 Mitarbeiterverwendung

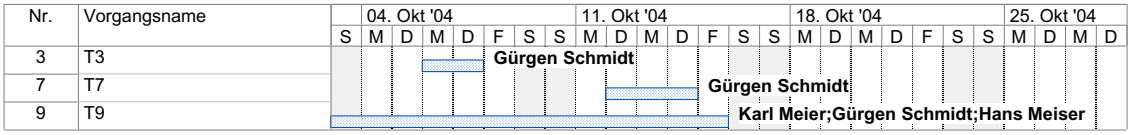

Abbildung 6: Mitarbeiterverwendung

Gibt es Spezialisten für bestimmte Aufgabenbereiche im Team, ist es zusätzlich sinnvoll, die Verwendung einzelner Mitarbeiter zu betrachten. Im Beispiel in Abbildung 6 wird die Verwendung des Mitarbeiters Gürgen Schmidt dargestellt. Erfasst werden alle Aufgaben, in die dieser Mitarbeiter involviert ist. Während T9 parallel zu den anderen Aufgaben durchgeführt wird, betreut Gürgen Schmidt T3 und T7 alleine. Verzögert sich T3 über die Pufferzeit hinaus, kann die Durchführung von T7 gefährdet werden, weil der alleinige Spezialist für diese Aufgabe nicht verfügbar ist, wodurch sich das Gesamtprojekt verzögert. Ein guter Projektplan sollte daher die Spezialistenaufgaben nicht im kritischen Pfad liegen haben.

4.3 Aufwandsschätzung

Die Schätzung des Aufwands für ein Softwareprojekt ist von *vielen* Faktoren abhängig. Vor allem in der Frühphase des Entwicklungsprozesses sind die genauen Anforderungen an das zu entwickelnde System häufig ebenso unbekannt oder vage, wie die zu verwendende Technologie zur Entwicklung oder die spätere Ausführungsumgebung. Daher kann auch die Aufwandsschätzung nur sehr ungenau erfolgen.

Daneben hängt der Aufwand für ein Softwareprojekt von der Produktivität des entwickelnden Teams ab, die sich u. a. aus den folgenden Faktoren ergibt (siehe [8], Seite 524):

Erfahrungen im Anwendungsgebiet Je mehr Erfahrungen für ein Anwendungsgebiet im Team vorhanden sind, desto effektiver ist die Softwareentwicklung in diesem Bereich. Je mehr Kompetenz ein Teammitglied auf einem Gebiet besitzt, desto wahrscheinlicher kann er seine Arbeit mit hoher Produktivität durchführen, da ihm beispielsweise Problemfelder bereits im Vorfeld bekannt sind.

Prozessqualität Je nach Wahl des Entwicklungsprozesses (siehe hierzu auch Kapitel 2) wird die Produktivität beeinflußt.

Projektgröße Große Projekte mit vielen Mitarbeitern erzeugen einen Kommunikationsoverhead innerhalb der Teams und mit den Kunden (die Anforderungen werden umfangreicher und müssen häufiger abgestimmt werden). Da in der Zeit, in der kommuniziert wird, keine aktive Entwicklungsarbeit vorgenommen wird, sinkt die Produktivität je Mitarbeiter.

Technische Unterstützung Der Einsatz neuer CASE-Tools wird die Produktivität zunächst senken, da der Umgang mit den neuen Werkzeugen zuerst erlernt werden muss. Auf der anderen Seite kann der Einsatz hochwertiger CASE-Tools, die eine gute Entwicklungsunterstützung bieten, zu einer Produktivitätserhöhung führen.

Arbeitsumgebung Ein gutes Teammanagement, wie es in Kapitel 3.1 beschrieben wird, führt zu einer Verbesserung der Produktivität der einzelnen Mitarbeiter. Dazu tragen auch ruhige Arbeitsumgebungen mit persönlichen Arbeitsbereichen bei.

Da die Schätzung der Projektkosten auf den Schätzungen des Projektaufwands beruht, setzen sich die Unsicherheiten auch für die Kosten fort. Sommerville ([8]) führt in Zusammenhang an, dass das Projektbudget zwar auf den Schätzungen beruht, das dann entwickelte Produkt aber in den meisten Fällen so *angepasst* wird, dass das Budget erfüllt wird (sowohl durch „Kürzungen“ am Produkt, als auch durch sinkende Produktivität zum Ausschöpfen des Budgets).

4.3.1 Methodenüberblick

Zur Kostenabschätzung von Softwareprojekten stehen unter anderem die folgende Methoden zur Verfügung (vgl. auch [10], Seiten 11f und [8]):

Algorithmische Aufwandsmodelle Als typischer Vertreter dieser Gattung von Aufwandsschätzungsmodellen gelten COCOMO (siehe Kapitel 4.3.3) und die Function Point Analyse (siehe Kapitel 4.3.2), die die Kosten mit Hilfe von Metriken, die aus Erfahrungswerten gebildet werden, berechnen.

Expertenbeurteilung Diese Methode greift auf das Wissen von Experten auf dem Gebiet der zu entwickelnden Software zurück. Die Experten liefern jeweils zunächst unabhängig voneinander ihre Schätzungen zu den Projektkosten ab. In einem zweiten Schritt werden die Schätzungen untereinander verglichen und diskutiert. Ein Ergebnis steht erst fest, wenn eine Einigung erreicht werden konnte, andernfalls werden erneut Einzelschätzungen vorgelegt.

Analogieschätzung Nur wenn in einem sehr ähnlichen Bereich bereits erfolgreich Projekte abgeschlossen wurden, kann dieses Verfahren angewendet werden. Die Kosten werden dann analog zum abgeschlossenen Projekt geschätzt.

Ergänzungen Daneben können die vorgestellten Verfahren durch *Top-Down* oder *Bottom-Up*-Vorgehen ergänzt werden. Beim *Top-Down*-Ansatz beginnt eine Schätzung auf der Systemebene und ermittelt die Gesamtfunktionalität. In weiteren Schritten werden die Aufwandsauswirkungen der darunter liegenden Funktionalität ermittelt. Dadurch finden z. B. Integrationsaufwand oder Konfigurationsmanagement Berücksichtigung.

Beim *Bottom-Up*-Vorgehen hingegen beginnt die Schätzung auf der Komponentenebene. Die Gesamtprojektkosten werden als Summe der Kosten für die Komponenten ermittelt. Entsprechend finden tiefer liegende Problemquellen innerhalb der Komponenten mehr Berücksichtigung als z. B. der Integrationsaufwand. Daher lässt sich konstatieren, dass beide Verfahren ihre spezifischen Vor- und Nachteile haben.

4.3.2 Function Point Analyse

Die Function Point Analyse ist eine Methode zur Ermittlung der Größe einer Software, die unabhängig von den verwendeten Technologien, Sprachen, Entwicklungsmethoden oder Hardwareplattformen ist, denn Function Points (FPs) messen ein System aus funktionaler Sicht. Durch die Veränderung von nicht-funktionalen Faktoren ändert sich lediglich der Aufwand, um einen konstanten Funktionsumfang zu gewähren. Daher bietet sich die FP Analyse an, um die Produktivität auch sehr unterschiedlicher, heterogener Systeme zu beurteilen.

Dahinter steht die Erkenntnis, dass sich beispielsweise der Umfang der Codezeilen von Assemblerprogrammen und einer höheren Programmiersprache nicht vergleichen lassen. Sehr wohl ließe sich aber der Funktionsumfang des Codes vergleichen. Damit ist die Methode einfach für nicht technisch versierte Nutzer zu verstehen (eine Java-Applikation zur Berechnung findet sich unter [7]).

Das Zählen von FPs ermöglicht es sogar, dass die verschiedenen Schritte im Softwareentwicklungsprozess wie Anforderungsdefinition, Entwurf, Implementierung oder Testen miteinander verglichen werden können. Anhand der Zunahme von FPs läßt sich im Projektverlauf genau verfolgen, in welchem Umfang sich Ergänzungen oder Änderungen ergeben haben. Ein starkes Wachstum von FPs ist daher ein Indikator für schlechte Kommunikation mit dem Auftraggeber oder innerhalb des Projektteams, da neue Funktionalität, die zunächst nicht bedacht wurde, in einem späteren Schritt hinzugefügt werden muss.

In die Ermittlung der FPs eines Programms fließen die folgenden Parameter ein (aus [8], Seiten 522ff):

- Externe Ein- und Ausgaben
- Benutzerinteraktionen

- Externe Schnittstellen
- Vom System verwendete Dateien

Alle Einzelwerte werden bezüglich ihrer Komplexität beurteilt und mit einem relativen Wert versehen. Die Vergabe dieser Werte sollte sich nach dem „Function Point Counting Practices Manual“ (derzeit in der Version 4.1) richten, das von der “International Function Point User Group“ (www.ifpug.org) veröffentlicht wird, um eine möglichst gute Vergleichbarkeit der Werte zu erreichen.

So ergibt sich der FP-Rohwert (UFC) aus der Häufigkeit eines Elements bestimmten Typs (H) und der Gewichtung der Komplexität (K):

$$UFC = \sum H \cdot K$$

Über eine Vielzahl weiterer Faktoren, wie dem Grad der Verteilung, der Leistungsfähigkeit, der Wiederverwendbarkeit, dem Installationsaufwand oder der Zahl der externen Schnittstellen, die von der Komplexität des Gesamtprojekts abhängen, wird mit Hilfe des UFC der Gesamtwert eines Projekts berechnet. Die genaue Berechnung wird beispielsweise in [4] erläutert.

Zwar ist die Bestimmung von Function Points auch von verschiedenen Personen durchführbar, diese sollten allerdings fachlich sehr kompentent, geschult und erfahren sein, damit die Komplexitätseinschätzungen möglichst vergleichbar vorgenommen werden. Die Abhängigkeit von den bewertenden Personen ist angesichts der Vielzahl von Parametern eines der Hauptprobleme des FP-Verfahrens.

4.3.3 COCOMO-Modell

Das COnstructive COst MOdel basiert ursprünglich auf den empirischen Erhebungen zu rund 60 Studien von Softwareprojekten, die einen Umfang von 2000 bis 100.000 Codezeilen (LOC: Lines Of Code) aufwiesen und in Programmiersprachen vom Assembler bis hin zu PL/I durchgeführt wurden (vgl. [11]). Aus den ermittelten Daten wurden Formeln konstruiert, die im Rahmen des erzeugten Modells eine Erklärung für die Beobachtungen lieferten.

Bis heute wurde das COCOMO in sehr vielen Projekten erfolgreich eingesetzt und im Laufe der Zeit weiter verfeinert. Das initiale „COCOMO 81“ bezog sich auf die Softwareentwicklung nach dem Wasserfallmodell (siehe Kapitel 2.1), bei dem der Anteil wiederverwendeter Software sehr klein ist. Durch den Wandel in den Entwicklungsmethoden im Software Engineering (z. B. Einsatz von CASE-Tools) und fehlender Berücksichtigung des Mitarbeitereinfluss (Motivation, Führung) ist COCOMO 81 heute nicht mehr so weit verbreitet (vgl. [8], Seite 530ff).

Das neuere COCOMO 2 unterscheidet z. B. nach den Software-Vorgehensmodellen, wie der Entwicklung von Prototypen, dem Komponentenansatz und dem Einsatz von Programmiersprachen unterschiedlichen Niveaus. COCOMO 2 erfasst dabei die Schätzungspräzision auf den verschiedenen Stufen der Softwareentwicklung, so dass spätere Schätzungen mit einem höheren Detailgrad in das Modell eingehen:

1. Die *frühe Prototypenstufe* nimmt eine Abschätzung auf Basis von Object Points mit Hilfe einer „einfachen“ (siehe [11]) Größen- und Produktivitätsformel vor. Damit ist COCOMO auch für die Aufwandschätzung von Projekten mit Prototypen und

die Nutzung vorhandener Komponenten geeignet. Der Grad der Wiederverwendung unter den Object Points und die Produktivität der Entwickler werden berücksichtigt.

Object Points sind dabei grob gewichtete Schätzungen der Faktoren: Anzahl der Bildschirmmasken, Anzahl der Berichte und Anzahl der Module in einer höheren Programmiersprache.

2. Die *frühe Entwurfsstufe* findet zeitlich nach der Spezifikation der Systemanforderungen und u. U. dem ersten Entwurf statt. Zur Schätzung wird auf Function Points zurückgegriffen, aus denen mit Hilfe von Tabellen die erwarteten Codezeilen (abhängig von der verwendeten Programmiersprache) berechnet werden. Auch hier wird zwischen vorliegendem oder generiertem und zu implementierendem Code unterschieden. Zusätzlich werden unter anderem der Aufwand für große Projekte, die Erfahrung auf dem Projektgebiet, der Teamzusammenhalt, die Produktkomplexität und nicht-funktionale Anforderungen wie Zuverlässigkeit aber auch der Integrationsaufwand berücksichtigt.

3. In der *Stufe nach dem Architekturentwurf* liegt bereits die Systemarchitektur vor, so dass die Schätzungen entsprechend genauer vorgenommen werden können. Auf dieser Stufe fließen auch Mitarbeiterqualifikationen, sowie konkrete Produkt- und Projekteigenschaften ein. In der Berechnung unterscheidet sich diese Stufe zum Einen in einer höheren Zahl von Faktoren, die berücksichtigt wird, zum Anderen wird die Quellcodeschätzung durch zwei Faktoren korrigiert:

   - Die *Unbeständigkeit der Anforderungen* entspricht der Anzahl an Codezeilen, die wahrscheinlich für eine Anpassung an neue Systemanforderungen zu modifizieren sind. Der Aufwand dieser Codezeilen wird zu der Gesamtschätzung addiert.
   - Der *Umfang einer möglichen Wiederverwendung* bildet einen umstrittenen Faktor. Er spiegelt die Anzahl der Zeilen Code wider, die wiederverwendet werden können. Dem COCOMO folgend, würden diese Codezeilen vom Gesamtaufwand subtrahiert. Da der Entwicklungsaufwand wiederverwendbarer Komponenten jedoch rund zwei mal so hoch ist (vgl. [8]), wie der „einfacher“ Komponenten, ist dieser Korrekturfaktor mit Vorsicht zu behandeln.

Insgesamt bietet auch das COCOMO 2-Modell, wie die Function Point Methode, eine Vielzahl von Parametern, die sehr viele Möglichkeiten zur „Fehlkonfiguration“ bieten. Damit bleiben sehr viele Unsicherheiten in der Abschätzung vorhanden. Bestehen für ein Projekt ausreichend eigene Daten aus vorangegangenen Entwicklungen, so sollte die Anwendbarkeit des COCOMO-Modells auf die eigenen Gegebenheiten anhand dieser Daten überprüft werden und gegebenenfalls zur Korrektur der Attributwerte verwendet werden, um die Schätzungen für die aktuellen Projekte präzisieren zu können.

5 Fazit

In allen im Text dargestellten Bereichen ist es von enormer Bedeutung, dass der Projektmanager möglichst erfahren und kompetent ist. Dies schlägt sich bei den Softwarevorgehensmodellen im Erkennen von Risiken und der richtigen Auswahl des Entwicklungsprozesses nieder. Im Bereich der Teamführung ist Erfahrung ebenso wichtig, um Probleme

im Team frühzeitig erkennen und beseitigen zu können. Ein guter Projektplan in Kombination mit richtiger Teamführung kann die Produktivität deutlich erhöhen und führt zu qualitativ hochwertigeren Produkten. Auch die Kosten- und Zeitschätzung wird ohne Erfahrungswerte ungenau und damit zu einem zusätzlichen Risiko für die Umsetzung eines Projekts. Damit ergeben sich hohe Anforderungen an den Projektmanager, der in den vorgestellten Modellen zu einer kritischen Schlüsselfigur wird.

Daneben sind vor allem in den Kosten- und Zeitschätzungsmodellen viele Unsicherheiten verborgen. Da beispielsweise im COCOMO 2-Modell eine Vielzahl von Parametern einfließen, kann eine auch nur minimale Abweichung aller Schätzungen in eine Richtung das Gesamtergebnis deutlich manipulieren, womit trotz Anwendung eines anerkannten Verfahrens nur unzureichende Vorhersagen erreicht werden können.

Prinzipiell gilt, dass mehr Erfahrung auch nur einer Person in einem Projektbereich von großem Vorteil für die Präzisierung der Schätzung von Kosten, Zeit und Risiken ist. Die Auswahl der richtigen und richtige Ausführung der Methoden, gleich in welchem der hier vorgestellten Bereiche, ist erst durch die Beschäftigung von Mitarbeitern mit praktischer Erfahrung auf einem Fachgebiet sinnvoll möglich. Diese Personen können über die Eignung von Verfahren für ein Projekt befinden und Projektteams bereichern.

6 Referenzen

Der Text wurde unter Verwendung der folgenden Literatur erstellt.

Literatur

[1] P. Feller, *Migration zu Windows 2000 - Leitfaden für effizientes Projektmanagement.* http://www.tellnettkomm.de/kapitel1.pdf: Springer Verlag, 1999.

[2] Q.-D. GmbH, *Quality Markup Language (QML)*, Q-DAS GmbH, http://www.qml-org.com, retrieved 2004-10-03.

[3] W. S. Humphrey and J. W. Over, *Introduction to the Team Software Process.* Addison Wesley Longman, Inc., Boston, San Francisco, September 2003.

[4] D. Longstreet, *Fundamentals of Function Point Analysis*, Longstreet Consulting Inc., www.SoftwareMetrics.Com, http://www.ifpug.com/files/Fundamentals of Function Point Analysis.pdf, 2004, retrieved 2004-09-25.

[5] P. N. Robillard, P. Kruchten, and P. d'Astous, *Software Engineering Process with the UPEDU.* Addison Wesley Longman, Inc., Boston, San Francisco, 2003.

[6] T. J. Scott, "Team selection methods for student programming projects," in *Proceedings of the 8th CSEE Conference*, New Orleans, 1995.

[7] smlab@irb.cs.uni–magdeburg.de, *Java Application Function Point Calculation*, Uni Magdeburg - SMlab, http://irb.cs.uni-magdeburg.de/sw-eng/us/java/fp/, retrieved 2004-09-25.

[8] I. Sommerville, *Software Engineering*, 6th ed. Pearson Studium, 2001, deutsche Ausgabe.

[9] P. Ulrich and E. Fluri, *Management*, 7th ed. UTB für Wissenschaft: Uni-Taschenbücher, 1995.

[10] K. Werdenich, *CoCoMO – COstructive COst MOdel*, Universität Salzburg – Institut für Computerwissenschaften, http://www.software-kompetenz.de/servlet/is/6818/COCOMO_Werdenich_Studarbeit.pdf ?command=downloadContent&filename=COCOMO_Werdenich _Studarbeit.pdf, 16.05.2002, retrieved 2004-09-25.

[11] Wikipedia, *CoCoMo*, Wikipedia - The Free Encyclopedia, http://en.wikipedia.org/wiki/COCOMO, 2004, retrieved 2004-09-25.

Verfasser der Arbeit

Klaus Krogmann
E-Mail: http://www.kelsaka.de/mail
Homepage: http://www.kelsaka.de